07

Creando la ConnectionFactory

Transcripción

[00:00] Hola, estamos avanzando con el proyecto y ya sabemos cómo abrir una conexión y cómo utilizarla para buscar registros en la base de datos. Tanto en la clase ProductoController cuando en la clase de PruebaConexión nosotros abrimos una conexión para probar que todo está bien, acá en PruebaConexión y en el ProductoController para devolver a la pantalla el listado de productos que tenemos en la base de datos, que están ahí registrados en nuestro depósito.

[00:26] Si paramos para ver mejor los códigos, vemos que en ambos casos acá, nosotros tenemos código repetido, me parece. Mira, tenemos el Connection con = DriverManager.getConnection, con los datos de conexión acá en el controller, y Connection con = DriverManager.getConnection, los datos de conexión en el PruebaConexión.

[00:49] Bueno, vamos ahí, replicando esta información acá. Tenemos, para tomar una conexión, ejecutamos la misma cosa dos veces, una en cada clase distinta. ¿Y qué podemos concluir aquí? Que si un día se cambia la URL de conexión de la base de datos o se cambia una cosa de la credencial del usuario, nosotros tenemos que acordarnos de que tenemos que hacer un cambio acá en PruebaConexión y tenemos que hacer un cambio también en ProductoController.

[01:19] O sea, tenemos que actualizar dos clases acá con los datos de conexión nueva. Bueno, y si creamos una clase más, que realiza conexiones con la base de datos o por ejemplo, tenemos acá el método de guardar, el método de

eliminar y modificar, nosotros vamos a estar poniendo otra vez la lógica para tomar la conexión.

[01:40] O sea, vamos a tener cada vez más otros puntos en donde vamos a estar utilizando este mismo método de conexión y son más puntos para acordarnos de actualizar cuando o si alguna cosa con la base de datos se cambia: una URL, una credencial de conexión.

[01:56] Bueno, o sea cuanto más crece la aplicación, más complejo va a ser para mantener esto replicado y también más riesgoso va a ser porque tenemos mucho riesgo de olvidarnos, de cambiar alguno de estos puntos y la aplicación va y puede romperse por eso.

[02:20] ¿Entonces acá qué podemos hacer para mejorar esta situación? Lo que vamos a hacer aquí es crear una clase que va a cuidar solamente de la creación de conexiones, entonces acá voy a hacer un New Class y esta clase va a llamarse CreaConexión. O sea, es una clase que solamente va a crear la conexión y acá vamos a crear un método, un public que devuelve una Connectión, el paquete Java.sql, que va a llamarse recuperaConexion.

[02:59] Ahí está, este es el método ¿y qué vamos a hacer con eso? Nosotros vamos a tomar esta lógica de conexión de acá, la voy a sacar de aquí y voy a agregar acá en el CreaConexión. Y en lugar de esta llamada voy a hacer a ver, Connection con = new CrearConexión().recuperaConexión(); ahí tengo que importar esta clase. Ahora sí, hago un new CreaConexión y ya él recupera conexión.

[03:39] Con eso yo tengo la conexión. Voy a copiar este pedazo y voy a hacer lo mismo acá con el PruebaConexión. Ahí con este el método, guardo todo y listo, pero acá tenemos que hacer el throws de SQLException porque es el error que nos puede pasar.

[03:58] Y ahora nos falta hacer que también devolver esta conexión entonces en lugar de asignar este comando de DriverManager.getConnection a una

variable, nosotros vamos a hacer un return DriverManager.getConnection. Listo, ya estamos.

[04:16] Bueno. En las otras clases, podemos organizar los imports para sacar lo que no estamos utilizando más. Y vamos a ejecutar estas clases para ver cómo está el resultado, cómo funciona, si funciona. Voy acá, hago un run en PruebaConexión y vamos a ver en la consola qué pasa: Cerrando la conexión.

[04:35] O sea funciona sin ningún problema. Voy a llamar acá el control de stock main. Voy a hacer un run as Java application también y vamos a esperar que se ejecute acá. Y también tenemos el resultado sin ningún problema. Sigue devolviendo los dos productos que tenemos en la base de datos.

[04:58] O sea, está todo perfecto, sigue funcionando y refactorizamos el código. Lo más interesante acá es que esta refactorización que realizamos es en realidad un design pattern, o sea un patrón de diseño llamado factory method. El factory method tiene como el objetivo encapsular el código de creación de un objeto específico, centralizando la lógica en un solo punto.

[05:23] La clase crea conexión. Entonces es una fábrica de conexiones. Entonces, siempre que la llamamos es porque queremos crear una conexión y este es uno de los varios ejemplos que tenemos de formas de utilizar este estándar. Vale la pena buscar un poco sobre este tema.

[05:41] Y para seguir el estándar vamos a refactorizar acá un poco más. Vamos a cambiar el nombre de esta clase, voy a hacer un clic derecho, refactor rename, y voy a llamarlo ConnectionFactory para que siga el estándar. Y también voy a refactorizar un poco más y la voy a mover de paquete.

[06:05] En lugar de estar acá en la raíz del paquete, voy a hacer también refactor, move. Y acá yo voy a crear un nuevo paquete llamado .factory. Hay algún finish, ahí está. Y tenemos ya nuestra ConnectionFactory adentro de su paquete correcto. Genial.

[06:27] Ahora estamos utilizando un patrón de diseño que encapsula toda nuestra lógica para crear conexiones con la base de datos. Vamos dando una forma más profesional para nuestra aplicación, haciendo lo mismo que los otros desarrolladores java ya hacen en el mercado.

[06:42] Y ahora que mejoramos el código, vamos a avanzar con las demás funcionalidades de la aplicación. Nos vemos en las próximas clases.